UCRL-ID-151968

Building CHAOS: an Operating System for Livermore Linux Clusters

Jim E. Garlick
Chris M. Dunlap

U.S. Department of Energy

Lawrence Livermore National Laboratory

February 21, 2002

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

Building CHAOS: an Operating System for Livermore Linux Clusters

Chris M. Dunlap, Jim E. Garlick
{garlick,cdunlap}@llnl.gov

February 21, 2002

1 Project Overview

The Livermore Computing (LC) Linux Integration and Development Project (the Linux Project) produces and supports the Clustered High Availability Operating System (CHAOS), a cluster operating environment based on Red Hat Linux. Each CHAOS release begins with a set of requirements and ends with a formally tested, packaged, and documented release suitable for use on LC's production Linux clusters.

One characteristic of CHAOS is that component software packages come from different sources under varying degrees of project control. Some are developed by the Linux Project, some are developed by other LC projects, some are external open source projects, and some are commercial software packages. A challenge to the Linux Project is to adhere to release schedules and testing disciplines in a diverse, highly decentralized development environment. Communication channels are maintained for externally developed packages in order to obtain support, influence development decisions, and coordinate/understand release schedules.

The Linux Project embraces open source by releasing locally developed packages under open source license, by collaborating with open source projects where mutually beneficial, and by preferring open source over proprietary software. Project members generally use open source development tools.

The Linux Project requires system administrators and developers to work together to resolve problems that arise in production. This tight coupling of operations and development is a key strategy for making a product that directly addresses LC's production requirements. It is another challenge to balance support and development activities in such a way that one does not overwhelm the other.

2 Scope

This report, written for Linux Project management and team members, describes a process that is intended to stabilize system software on existing and future LC Linux clusters. The discussion is confined to the process of designing, building, and testing a new CHAOS release. The report does not address the details of specific tools or development systems used by the project.

CHAOS will initially run on the two PCR (Parallel Capacity Resource) Intel/Quadrics-based Linux clusters installed in Q3CY2001. A third, larger cluster of similar architecture named MCR (Multiprogrammatic and Institutional Computing [M&IC] Capability Resource) and anticipated in Q3CY2002 will also be supported. Schedules for supporting future Linux hardware variations will be determined as LC's Linux plans evolve.

The PCR systems currently run system software that was installed and tested using ad hoc methods during acceptance testing. This software has evolved to near production readiness through triage over the course of 6 months. While the install/triage approach gets results quickly, the level of effort required to maintain stability is not sustainable, and the end product will always be fragile without formalized configuration management and defect tracking. For example, an operation as simple as reinstalling the software from scratch requires the participation of everyone who may have ever installed "emergency fixes" during triage and that everyone's memory and notes are accurate.

The methods outlined in this report should result in stable, supportable system software that can be maintained and improved with a sustainable effort.

3 Software Life Cycle

Figure 1 shows the CHAOS life cycle. The phases leading up to each release are requirements definition, development, unit testing, staging, integration testing, and packaging. Multiple releases may be in the pipeline simultaneously; for example, while one release is staging, the next release may be in requirements definition. Releases should occur about every 6 months.

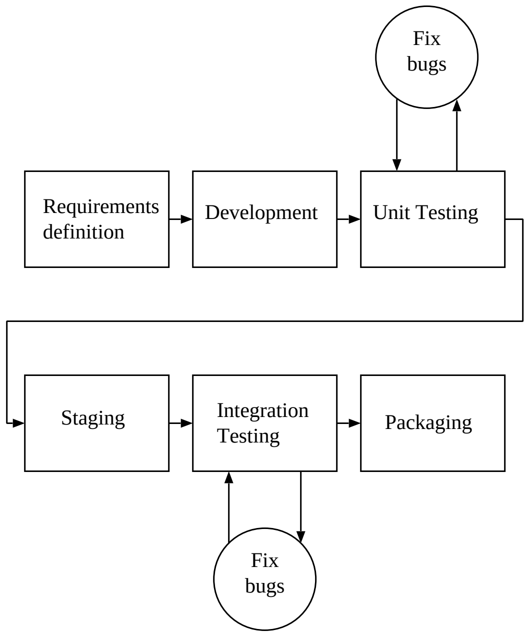

Figure 1: CHAOS life cycle.

Each software package included in a release has an "owner" who carries that package through each phase of the CHAOS life cycle. A package developed within the Linux Project is owned by the lead developer of that package. Software packages developed by other LC projects may be owned by a Linux Project member or one of the package developers, depending on the level of Linux support negotiated with the project. External open source and commercial software packages are owned by a Linux Project member who is responsible for understanding external release schedules and maintaining a channel of communication with appropriate companies and/or individuals.

Some software packages may need to be updated at times other than the normal CHAOS release cycle. Such non-synchronized updates will be restricted to software components interacting with outside systems (such as DPCS or HPSS) or to critical bug fixes as described in Sec. 4, and they will undergo testing under the direction of the component's owner.

3.1 Requirements Definition Phase

New CHAOS releases are driven by the need for enhanced functionality and bug fixes and by the desirability of staying reasonably current with new versions of component software. Practically speaking, it is likely that a new release of CHAOS will follow each minor Red Hat Linux release (e.g., the 7.1 to 7.2 transition).

The requirements definition phase is a project planning process led by the project manager. Input comes from the package maintainers, users, LC management, project leaders, and the defect tracking system, which tracks both defects and change requests. The result is a release plan document, prepared by the project manager, that describes the release contents and delivery schedule and reflects mutual agreement among the stakeholders.

The release plan includes the following:

- A table of packages, their major versions, and owners.
- A list of important new functionality.
- A list of defects to be corrected.
- A schedule for release.

3.2 Development Phase

Development commences once requirements have been established. During this phase, package owners work independently to meet deliverables according to the release plan. Internally developed software is subject to the project development guidelines described in Sec. 5.

Externally developed packages follow their own guidelines and are expected to be developed out of sync with CHAOS. The role of an external package owner during the development phase is to ensure that CHAOS issues (such as outstanding bugs or requests for new functionality) are addressed and to communicate any deviations in schedule that affect CHAOS release plans to the project manager. External package owners may also be responsible for developing unit tests and packaging strategies.

All package owners will keep releases under revision control (even for binary-only distributions), repackage them in Red Hat Package Manager (RPM) format if necessary, develop a suite of automated unit tests, and track bugs using the project defect tracking system.

3.3 Unit Testing Phase

Unit tests verify that a package functions correctly by itself. Each package owner develops a suite of automated tests that exercise all code paths of the software package to the extent possible. Ideally, these tests are developed in parallel with the package in order to test each new feature as it is implemented; however, unit tests for externally developed packages may have to be written after the fact.

Automated unit tests should be packaged in such a way that they can be rerun on the target operating environment.

A package is ready for staging when all associated items in the release plan document have been addressed and the package has passed all of its unit tests.

3.4 Staging Phase

In the staging phase, a development system is installed from scratch with all packages (including the core Red Hat Linux distribution) that make up the CHAOS release in preparation for integration testing.

Package interdependences determine the order in which some packages are staged; for example, the Red Hat operating system and Quadrics-enabled kernel must be installed before the RMS batch system. The release manager determines an order for this process.

All packages installed during staging are in RPM format and are assigned a version number that does not change in the final CHAOS unless bugs are found during integration testing.

Once all components are installed, the release is frozen as integration testing begins; future package updates are performed only under direction of the release manager.

3.5 Integration Testing Phase

Integration testing is overseen by a release manager who is responsible for ensuring the quality of a release and for documenting the integration test process. The release manager develops tests in association with the owners of various packages in order to exercise interactions between packages, schedules open-minded users to volunteer to run real workloads on the system in the latter stages of testing, and locates workloads, test suites, and regression tests to include in the integration test phase.

An example of a trivial integration test is submitting a "hello world" MPI job to DPCS. This test exercises DPCS, SLURM, the MPI libraries, and the Quadrics interconnect in combination. Integration testing should include regression tests designed to re-create problematic situations from previous releases, as recorded in the defect tracking system.

To the extent possible, tests are automated and run from a common test framework. Some tests are packaged as an RPM to be installed as part of the final release; this allows these tests to be run on a new hardware installation for validation. Time should be allocated following a production CHAOS update to allow the release manager to run these tests before the machine is made available to users.

When problems are discovered in the integration test phase, the release manager calls on the owners of relevant packages to assist with debugging. Once problems are corrected, updated packages are installed and integration testing resumes. All bugs discovered in this phase should be documented in the defect tracking system, and any package updates should be accompanied by a change in the RPM release number for that package. Bugs may result in the addition of new regression tests. The release manager carefully documents and controls package updates and (re-)execution of integration tests. When all known bugs are fixed, it is time to introduce a release.

3.6 Packaging Phase

The release manager is also responsible for packaging CHAOS. Each release consists of the following:

- A set of RPM and SRPM files containing the components of the release, including unit and integration test scripts.
- A manifest containing the list of components and individual versions.
- Release notes.
- An installation guide.

Each release is stored on the project file server (and is therefore written to the backup archive) and written to CDR media.

Before the release is distributed, the development cluster is reinstalled from the CDR media, following the instructions in the installation guide. Any necessary fine tuning of the installation guide or release notes can occur at this point.

When everything is in order, a set of CDR images is created and distributed to system administrators for installation on production systems. As described in Sec. 3.5, system tests may be repeated after installation in order to verify the release on the actual hardware, especially if the hardware is significantly larger or otherwise different from the development system used to test the release.

4 Software Support

Unlike some Linux cluster efforts, CHAOS is not a research project; it is a production-focused development project. All team members have both development and support responsibilities.

When a problem arises on a production system, first-order problem diagnosis is performed by the system administrators. If they determine that the problem is related to CHAOS, they enter it into the LC trouble ticket system for further analysis by the development team. In some cases, developers may be required to assist with the initial diagnosis.

When a problem is determined to be a bug in CHAOS, it is entered into the project defect tracking system. If the project management team determines that the defect is urgent enough, it can be fixed immediately, and an updated version[1] of the package made available via the project web site. The procedure for testing bug fixes is not formalized; it could range from executing the entire integration test suite to running just the package's unit tests. Because testing requirements at this stage are less rigorous, it is desirable to defer bug fixes to the next release of CHAOS; when that is not possible, the fix should be applied directly to the version exhibiting the bug (via a branch in the revision control system) in order to avoid inadvertently introducing new bugs into production due to untested functionality. The bug remains open in the defect tracking system until the fix has been applied to all branches of development that lead to future releases.

5 Project Software Development Practices

As previously discussed, some components of CHAOS are internally developed. These packages must conform to project development procedures.

New packages, subsystems, or major functionality being developed by the project undergoes a design review in which the primary developer presents a detailed design to the project team. A set of slides and/or a document of the design detail is placed on the project web server. Development begins once the project team is satisfied that the design is reasonable and that it meets the established requirements.

Software will be developed using good coding practices (Maguire, 1993)(McConnell, 1993). There is not yet a formalized project style guide, but coding and documentation style should be internally consistent and readable by others. Reasonable inline documentation is encouraged (e.g., briefly describe inputs and outputs of each function and any tricky spots in the code).

The project revision control system will be used to track all project software. Except in the very early stages of a project, committed changes should be fully debugged. The revision control system can be configured to send e-mail whenever code is checked in, thereby serving as a rudimentary status report to interested parties.

Informal code walkthroughs (Wiegers, 2002) will be conducted as appropriate when coding milestones are achieved. An informal code walkthrough is a scaled-down, less time-consuming version of the formal code review. The walkthrough is intended to identify coarse defects, not to scrutinize every line of code. The author informally walks a small group of team members through the code at the module or function level. Unit tests are also reviewed. At the end of a code walkthrough, the author should carry away a list of defects in both the code and the unit test strategy, and the group should have reached a shared understanding of the code's purpose and implementation. This understanding strengthens the group's ability to share the support workload and promotes sharing of code and programming techniques between individuals.

[1]The RPM version number is updated along with the software, unlike IBM's EFIX mechanism.

References

Maguire, S.: 1993, *Writing Solid Code: Microsoft's techniques for developing bug-free C programs*, Redmond, WA: Microsoft Press

McConnell, S.: 1993, *Code Complete: a practical handbook of software construction*, Redmond, WA: Microsoft Press

Wiegers, K. E.: 2002, *Peer Reviews in Software: a practical guide*, Boston: Addison-Wesley